Visualization Using SciPy and Seaborn

CS 3753 Data Science

Prof. Weining Zhang

Topics

In [ ]:
```
%matplotlib inline

import numpy as np
import pandas as pd
from scipy import stats, integrate
import matplotlib.pyplot as plt

import seaborn as sns
sns.set(color_codes=True)

np.random.seed(sum(map(ord, "distributions")))
```

Seaborn

Seaborn (https://seaborn.pydata.org/) is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. It includes the following features.

- Style functions
- Color palettes
- Distribution plots
- Categorical plots
- Regression plots
- Axis grid objects

Visualize a Relationship Involving Categorical Data

- Categorical scatterplots:

```
stripplot() (with kind="strip"; the default)
swarmplot() (with kind="swarm")
```

- Categorical distribution plots:

```
boxplot() (with kind="box")
violinplot() (with kind="violin")
boxenplot() (with kind="boxen")
```

- Categorical estimate plots:

```
pointplot() (with kind="point")
barplot() (with kind="bar")
countplot() (with kind="count")
```

Example: Draw a Categorical Plot onto a FacetGrid

```
factorplot(x=colName, y=colName, hue=colName, data=df,...)
```

- Placement control

```
- col=colName
- col_wrap=n    : n subplot in a row
- size=height
- aspect=r      : width:height ratio
```

In [ ]:
```
sns.set(style="ticks")
exercise = sns.load_dataset("exercise")
exercise[:10]
```

In [ ]:
```
g = sns.factorplot(x="time", y="pulse", hue="kind", data=exercise)
```

In [ ]:
```
titanic = sns.load_dataset("titanic")
titanic[:10]
```

In [ ]:
```
g = sns.factorplot("alive", col="deck", col_wrap=5,
                   data=titanic[titanic.deck.notnull()],
                   kind="count", size=2.5, aspect=.8)
```

Example: Plot Pairwise Relationships in a Dataset

```
pairplot(df)
```

- One scatter subplot for each pair of columns
- Display $n \times n$ grid layout
- Display the univariate distribution of each column on the diagonal

In [ ]:
```
iris = sns.load_dataset("iris")
iris[:10]
```

In [ ]:
```
sns.pairplot(iris);
```

In [ ]:
```
g = sns.pairplot(iris, hue="species", palette="husl")
```

Much like the relationship between jointplot() and JointGrid, the pairplot() function is built on top of a PairGrid object, which can be used directly for more flexibility:

In [ ]:
```
g = sns.PairGrid(iris)
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot, cmap="Blues_d", n_levels=6);
```

Plotting Univariate Distributions

```
distplot(x, bins=..., hist=..., kde=..., rug=..., )
```

- By default, draw a histogram and fit a kernel density estimate (KDE)
- hist, kde and rug are binary parameters

In [ ]:
```
x = np.random.normal(size=100)
#print(x)
sns.distplot(x);
```

Histograms

Histograms are likely familiar, and a hist function already exists in matplotlib. A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.

To illustrate this, let's remove the density curve and add a rug plot, which draws a small vertical tick at each observation. You can make the rug plot itself with the rugplot() function, but it is also available in distplot():

In [ ]:
```
sns.distplot(x, kde=False, rug=True);
```

When drawing histograms, the main choice you have is the number of bins to use and where to place them. distplot() uses a simple rule to make a good guess for what the right number is by default, but trying more or fewer bins might reveal other features in the data:

In [ ]:
```
sns.distplot(x, bins=30, kde=True, rug=False);
```

Kernel Density Estimaton

- The KDE plots encodes the density of observations on one axis with height along the other axis
- KDE process is based on estimation of normal distribution and may be complex
- The tightness of estimation can be controled by a bandwidth parameter bw
- The spread of the sides can be controled by the cut parameter

In [ ]:
```
sns.distplot(x, hist=False, rug=True);
```

The bandwidth (bw) parameter of the KDE controls how tightly the estimation is fit to the data, much like the bin size in a histogram. It corresponds to the width of the kernels we plotted above. The default behavior tries to guess a good value using a common reference rule, but it may be helpful to try larger or smaller values:

In [ ]:
```
sns.kdeplot(x)
sns.kdeplot(x, bw=.02, label="bw: 0.2")
sns.kdeplot(x, bw=2, label="bw: 2")
plt.legend();
```

The nature of the Gaussian KDE process means that estimation extends past the largest and smallest values in the dataset. It's possible to control how far past the extreme values the curve is drawn with the cut parameter; however, this only influences how the curve is drawn and not how it is fit:

In [ ]:
```
sns.kdeplot(x, shade=True, cut=0.5)
sns.rugplot(x);
```

Fitting Parametric Distributions

- We can use distplot() to fit a parametric distribution to a dataset and visually evaluate how closely it corresponds to the observed data

In [ ]:
```
x = np.random.gamma(6, size=200)
sns.distplot(x, kde=True, fit=stats.gamma, rug=True);
```

Plotting Bivariate Distributions

```
jointplot(x=col, y=col, kind=..., ...)
```

- Creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes
- The kind of plots include scatter plot, kde plot, hexbin plot, etc.
- We can also
  - use kdeplot() with matplotlib Axes object
  - work with JointGrid directly

In [ ]:
```
mean, cov = [0, 5], [(1, -.5), (-.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
df[:10]
```

Example: Scatterplots

In [ ]:
```
sns.jointplot(x="x", y="y", data=df);
```

Example: Hexbin Plots

- Each bin shows the counts of observations that fall within a hexagonal bin.
- This plot works best with relatively large datasets.
- It looks best with a white background.

In [ ]:
```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
```

Example: Kernel Density Estimation

- This plot is shown with a contour plot and is available as a style in jointplot().

In [ ]:
```
sns.jointplot(x="x", y="y", data=df, kind="kde");
```

Example: Use kdeplot() Directly on Axes

- Control the placement of each two-dimensional kernel density plot by placing each on a specific (and possibly already existing) matplotlib axes.

In [ ]:
```
f, ax = plt.subplots(figsize=(6, 6))
sns.kdeplot(df.x, df.y, ax=ax)
sns.rugplot(df.x, color="g", ax=ax)
sns.rugplot(df.y, vertical=True, ax=ax);
```

- If you wish to show the bivariate density more continuously, you can simply increase the number of contour levels:

In [ ]:
```
f, ax = plt.subplots(figsize=(6, 6))
cmap = sns.cubehelix_palette(as_cmap=True, dark=0, light=1, reverse=True)
sns.kdeplot(df.x, df.y, cmap=cmap, n_levels=50, shade=True);
```

Example: Work With JointGrid

- jointplot() returns the JointGrid object after plotting, which can be used to add more layers or to tweak other aspects of the visualization

In [ ]:
```
g = sns.jointplot(x="x", y="y", data=df, kind="kde", color="m")
```

In [ ]:
```
g.plot_joint(plt.scatter, c="w", s=30, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0)
g.set_axis_labels("$X$", "$Y$");
g
```

Visualize Regression Models

Seaborn provides a visual guide to show patterns in a dataset during exploratory data analyses.

- `regplot(x=colName, y=colName, data=df, ...)`
- `lmplot(x=colName, y=colName, data=df, ...)`
- `xxxplot(.... kind="reg")`

- lmplot() is more flexible than regplot()

Example

Plot linear model using tips data set

- a scatter plot
- a linear model
- a 95% confidence interval

In [ ]:
```
tips = sns.load_dataset("tips")
sns.regplot(x="total_bill", y="tip", data=tips)
```

In [ ]:
```
sns.lmplot(x="total_bill", y="tip", data=tips)
```

Example: Linear Model with Categorical Predictor

When the x is categorical, the scatter plot is not optimal.

The solution include:

- add some random jitters
- use some statistics over each bin

In [ ]:
```
sns.lmplot(x="size", y="tip", data=tips, x_jitter=.05)
```

In [ ]:
```
sns.lmplot(x="size", y="tip", data=tips, x_estimator=np.mean);
```

Example: A Case Where A Liner Model is not the Best Fit

In [ ]:
```
anscombe = sns.load_dataset("anscombe")
```

In [ ]:
```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           ci=None, scatter_kws={"s": 80});
```

In [ ]:
```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           order=2, ci=None, scatter_kws={"s": 80});
```

Example: Plot Regression Models with Multiple Response Variables

- Additional categorical response variables can be used to seperate plots of y and x

In [ ]:
```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips);
```

In [ ]:
```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
           markers=["o", "x"], palette="Set1");
```

In [ ]:
```
sns.lmplot(x="total_bill", y="tip", hue="smoker", col="time", data=tips);
```

In [ ]:
```
sns.lmplot(x="total_bill", y="tip", hue="smoker",
           col="time", row="sex", data=tips);
```

Example: Plot Regression Models in Other Plotting Context

In [ ]:
```
sns.jointplot(x="total_bill", y="tip", data=tips, kind="reg");
```

In [ ]:
```
sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"],
             size=5, aspect=.8, kind="reg");
```

In [ ]:
```
sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"],
             hue="smoker", size=5, aspect=.8, kind="reg");
```